Smart phone & Cross-platform Communication Toolkit User Manual for Android

Release 2.1.0

June 2012 Edition

Worldwide technical support and product information:

www.toolsforsmartminds.com

TOOLS for SMART MINDS Corporate headquarter

Via Padania, 16 Castel Mella 25030 Brescia (Italy)

Copyright © 2010 Tools for Smart Minds. All rights reserved.

Contents

Figure index 6

ABOUT THIS MANUAL 7

Conventions 7

INTRODUCTION 8

Overview 8

Top reasons to use Smartphone & cross platform Communication Toolkit (SCCT) 8

SCCT application examples 9

Communicating data across a network, on a complex assembly line 9

Publishing real time data to mobile devices 9

Delivering high quality maintenance service to your customer 9

Supported platforms 9

publisher library 9

Subscriber library 10

INSTALLATION 11

Install SCCT Subscriber on Eclipse 15

GENERAL WORKING 17

HOW SCCT JAVA SUBSCRIBER WORKS? 18

HOW THE PACKAGES ARE MANAGED 19

Permission 19

QUICK START 21

ERRORS 25

FIGURE INDEX

FIGURE 1 - NETBEANS TOOLS MENU 11

FIGURE 2 - LIBRARY MANAGER WINDOW ON NETBEANS 12

FIGURE 3 - NEW LIBRARY WINDOW ON NETBEANS 12

FIGURE 4 - BROWSE JAR/FOLDER WINDOW ON NETBEANS 13

FIGURE 5 - PROJECT MENU ON NETBEANS 13

FIGURE 6 - PROJECT PROPERTIES WINDOW ON NETBEANS 14

FIGURE 7 - ADD LIBRARY WINDOW ON NETBEANS 14

FIGURE 8 - PROJECT MENU ON ECLIPSE 15

FIGURE 9 - PROJECT PROPERTIES WINDOW ON ECLIPSE 16

FIGURE 10 - JAR SELECTION WINDOW ON ECLIPSE 16

FIGURE 11 - UML SCHEME OF OBSERVER PATTERN 18

FIGURE 12 - ANDROIDMANIFEST 20

FIGURE 13 - NEW JAVA CLASS MENU ON ECLIPSE 21

FIGURE 14 - CONCRETEOBSERVER CLASS EXTENDS SCCT_OBSERVER CLASS 21

FIGURE 15 - IMPORT SCCT.JAVASUBSCRIBER.SCCT_OBSERVER 21

FIGURE 16 - UNIMPLEMENTED METHODS OF SCCT_OBSERVER 22

FIGURE 17 - IMPORT SCCT CLASS 22

FIGURE 18 - IMPLEMENTED METHODS OF SCCT_OBSERVER 23

FIGURE 20 – MAIN ACTIVITY CLASS IMPLEMENTATION 23

FIGURE 21 - IMPORT SCCT_SUBJECT CLASS 23

FIGURE 22 - REGISTER OBSERVER 23

FIGURE 23 - OPEN COMMUNICATION AND START DATA 24

FIGURE 24 - FINAL RESULT 24

About this Manual

The Smartphone & Cross-platform Communication Toolkit User Manual describes the virtual instruments (VIs) used to communicate and pass data between LabVIEW and either a local or remote application. You should be familiar with the operation of LabVIEW, your computer and your computer operating system.

Conventions

The following conventions appear in this manual:

▸	The ▸ symbol leads you through nested menu items and dialog box options to a final action. The sequence **Tools** ▸ **Options** directs you to pull down the **Tools** menu, select **Options** item.
Bold	Bold text denotes items that you must select or click on the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

`monospace italic`

Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

INTRODUCTION

This chapter describes the installation procedure, installed components, and the main features of the Smartphone & Cross-platform Communication Toolkit.

OVERVIEW

The Smartphone & Cross-platform Communication Toolkit is an add-on package for communicating data trough applications. The toolkit contains a set of high level functions for sending your application data and advanced functions for customized tasks.

The following list describes the main features of the Smartphone & Cross-platform Communication Toolkit:

- Works over any TCP/IP connection
- Works over Local Area Networks as well as Internet connections.
- Implements the publisher – subscriber pattern (also known as Observer pattern)
- Authenticates subscribers through a API-KEY.
- Controls in background the state of every connection to identify loss of communication.
- Publishes GPS coordinates to manage mobile systems.
- Works with platform independent Data format and communicate with multiple platforms at the same time: third party vendors have implemented toolkit to develop on Android platform, Java, .NET and VB, Unix and iOs.

TOP REASONS TO USE SMARTPHONE & CROSS PLATFORM COMMUNICATION TOOLKIT (SCCT)

Adopting this toolkit you have the following advantages:

Simplify communication: don't care about communication details over a TCP communication channel, SCCT does it for you.

Multiple platforms are supported: exchange your data with a protocol supported on a wide range of platforms and programming languages.

It's reliable: many applications have been created with this toolkit around the world.

Speed up your development activity: this toolkit allows the creation of distributed application and let you save a lot of your time.

SCCT has been created by LabVIEW developers for LabVIEW developers: it includes some great features supported on LabVIEW platform only (see Publishing CustomData in this manual) so if you need to exchange data with other LabVIEW applications, take advantage of the power of SCCT to deliver high quality code and reduce developing time.

Today applications need to retrieve information from database, bar code readers, OCR systems, remote data acquisition or technical operators with mobile devices. Often you have to create systems capable to exchange data with legacy applications created on different platforms. Every time you have to design an ad hoc communication protocol, code both server and client side routines, debug them. Every time! With SCCT you have a new tool that let

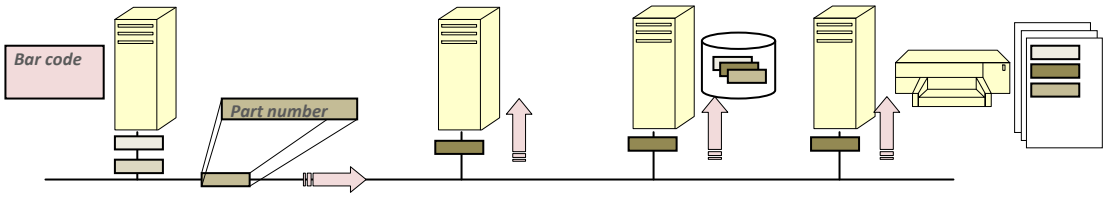
you save a lot of time, lets you focus on your project's core and manage multiple communications at the time. In the following examples you are going to see how SCCT can dramatically improve the quality of your software solutions.

SCCT APPLICATION EXAMPLES

SCCT can be successfully applied in many real world situations. In this chapter we discover where SCCT helps developers to succeed to deliver high valued solutions.

COMMUNICATING DATA ACROSS A NETWORK, ON A COMPLEX ASSEMBLY LINE

Creating an application which broadcasts part numbers and barcodes coming from databases or bar code readers, to all computers on a assembly line, as schematized below. On modern assembly lines, many computers control single processing stations and they need to exchange data among them to know, part numbers, print codes and certificates, store test results on different databases. SCCT provides many functionality to exchange data with applications developed with heterogeneous programming languages. besides, when your customer asks you to show some data on a mobile device, SCCT is capable to communicate with a large variety of smart phones and tables and you don't have to change a single line of code of your software. SCCT cares of all connected devices and communicate your data to all of them at the same time, receives user's requests and organize them in a time-ordered FIFO so that you can process them easily.



PUBLISHING REAL TIME DATA TO MOBILE DEVICES

When you have to publish acquired data in real time, SCCT is the best choice because with few SCCT Vis your applications is supported on an wide range of platforms: Android, iPhone, etc. SCCT is safe because every device must authenticate to your application and you can control who is connected in any time. Using SCCT you deliver open solution to your customers because they can use their favorite mobile platform to connect to your data. And if customers change mobile device family, you don't need to modify your code!

DELIVERING HIGH QUALITY MAINTENANCE SERVICE TO YOUR CUSTOMER

Including SCCT into your existing applications, let you offer fast support to your customers. You and your customers can monitor deployed applications everywhere, with a tablet, phone, desktop. integrating SCCT capabilities into existing applications makes easy debug and signal analysis of deployed systems around the world.

SUPPORTED PLATFORMS

SCCT is composed by two main components:

- publisher library
- subscriber library

PUBLISHER LIBRARY

this library let you create a full-featured publisher, which authenticates incoming subscribers, check connection status, sends data to all active publishers and passes their request to your application.

This library is available as a set of Vis for LabVIEW 2010 or later.

To get more details or download an evaluation copy of this library please visit:

http://www.toolsforsmartminds.com/products/SCCT.php

SUBSCRIBER LIBRARY

This library let you create a subscriber which handles all communication details with a publisher so you don't have to. It receives data packages and present them to your application according to their data types.

This library is available for the following platforms and languages:

Name	***Operating System***	***Development Language***
SCCT Subcriber for LabVIEW[1]	Windows	LabVIEW 2010
SCCT Subscriber for VB[2]	Windows	Visual Basic, .NET,
SCCT Subscriber for Java	Java VM 5.0 or later	Java
SCCT Subscriber for Android	Android 2.1 or later	Java
SCCT Subscriber for Linux[3]	Linux Kernel 2.6.30	ANSI - C
SCCT Subscriber for iPhone/iPad[4]	iOs	Objective C
SCCT Subscriber for Phone7[5]	Phone7	.NET

To get more details or download your **free** copy of SCCT subscriber library, please visit:

http://www.toolsforsmartminds.com/products/SCCT.php

[1] This library is included in SCCT VIPM package and is not available separately from SCCT Publisher library.

[2] This library is distributed as OCX component and can be used by any language that support ActiveX technology.

[3] This library will be available from September 2011. this library works on x86 32bit and ARM ver. 11 processor families

[4] This library will be available from Q4 2011.

[5] This library will be available from Q4 2011.

INSTALLATION

You can download all version of SCCT subscriber library from http://www.toolsforsmartminds.com/products/SCCT.php

Install SCCT Subscriber on NetBeans

First of all you need to NetBeans on your computer (it is possible download it free from http://netbeans.org/).

Once installed it, you can follow next steps:

1. Open NetBeans.
2. Open **Library Menager** window from **Tools>Libraries** menu.

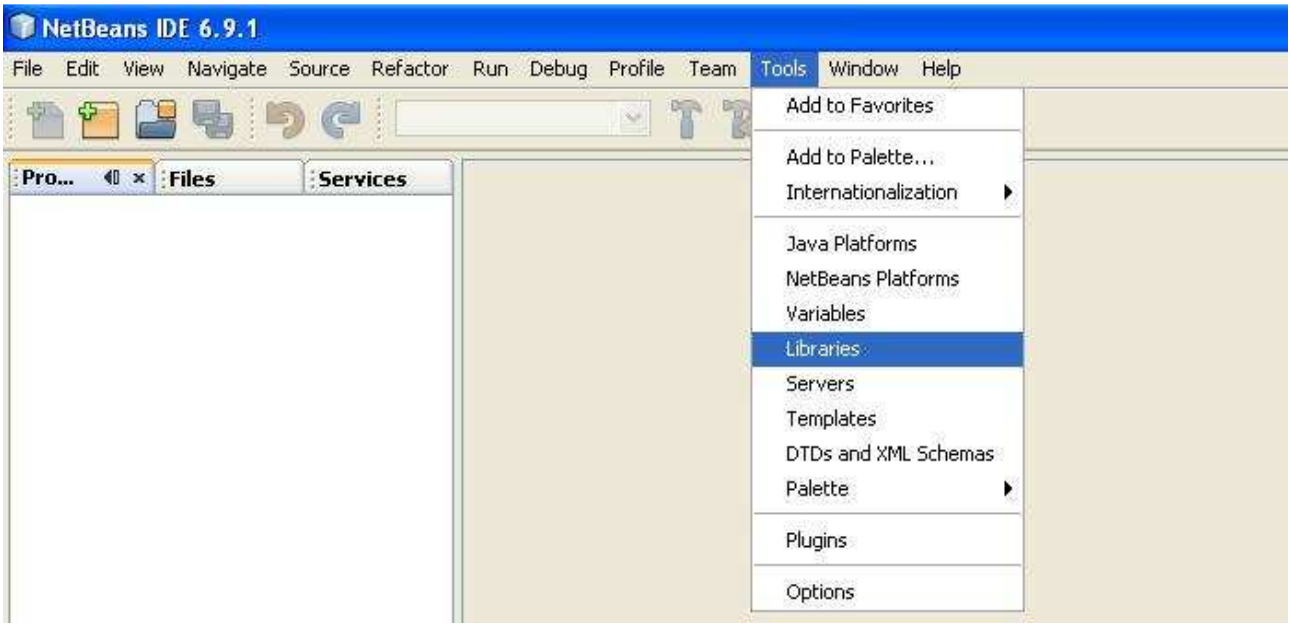


FIGURE 1 - NETBEANS TOOLS MENU

3. Click on New Library...

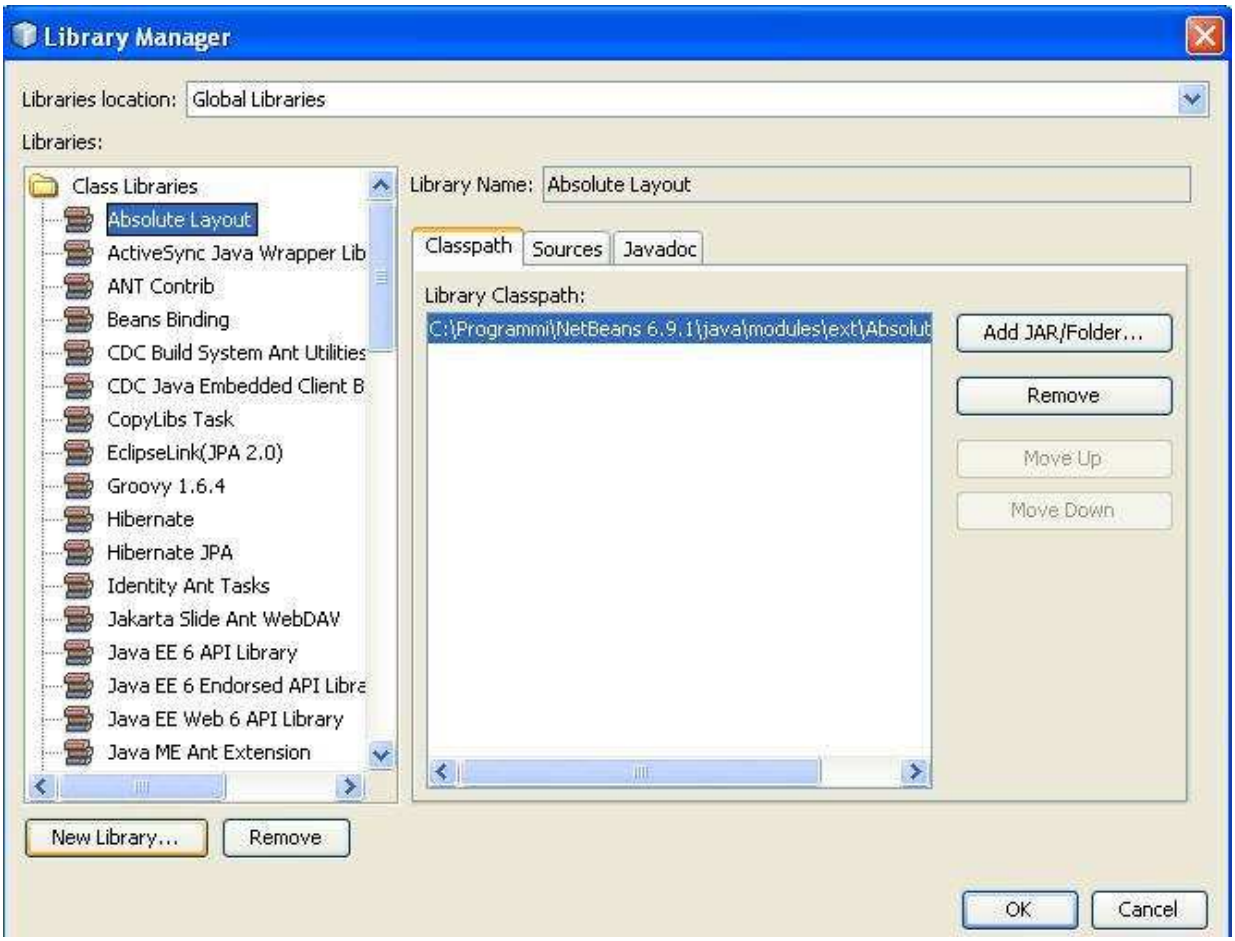


FIGURE 2 - LIBRARY MANAGER WINDOW ON NETBEANS

4. Insert "SCCT subscriber for Java" into the field and confirm by clicking **OK**.

FIGURE 3 - NEW LIBRARY WINDOW ON NETBEANS

5. Keeping selected the just insert library, in the classpath section click on **Add Jar/Folder…** and select the downloaded version of library indicating the path.

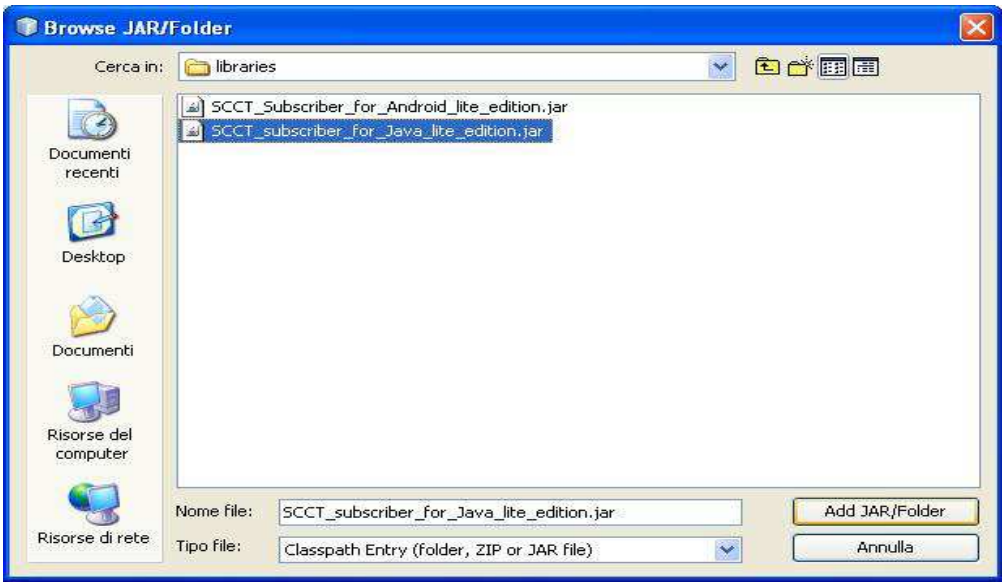


FIGURE 4 - BROWSE JAR/FOLDER WINDOW ON NETBEANS

6. Click **Ok** to close the window.

Now, every time you want use the library you have to add it to the project by mean of the following steps:

- Right click mouse on the project in which you want add it. From drop-down menu click **Properties.**

FIGURE 5 - PROJECT MENU ON NETBEANS

- From **Project Properties** window, select **Libraries** from **Categories** list

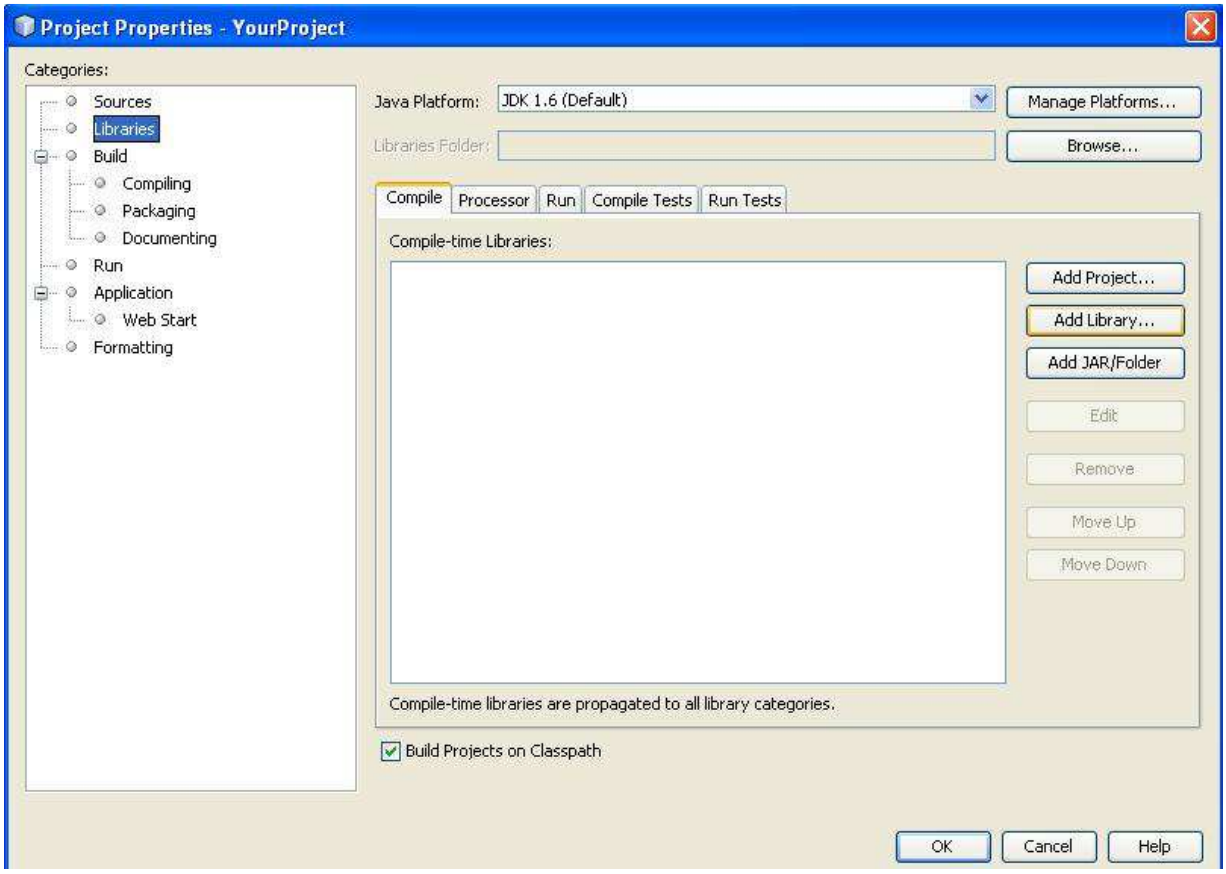


FIGURE 6 - PROJECT PROPERTIES WINDOW ON NETBEANS

- In the **Compile** section, click on **Add Library…** and add SCCT subscriber library

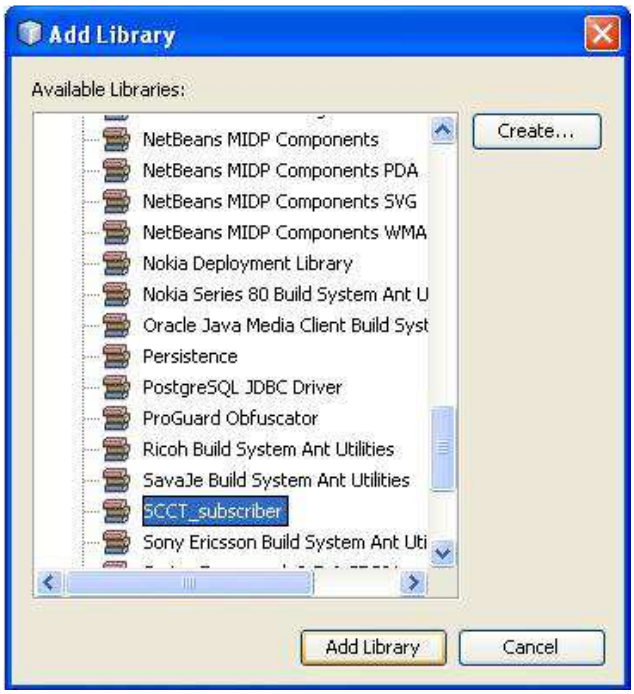


FIGURE 7 - ADD LIBRARY WINDOW ON NETBEANS

- Click **OK** to confirm and close the window.

INSTALL SCCT SUBSCRIBER ON ECLIPSE

Also in this case you can download a free version of Eclipse from http://www.eclipse.org/downloads/ .

If you want use the version for Android you also have to install the Android SDK. You can find a useful guide on : http://developer.android.com/sdk/installing.html .

To add the library on the project you can do:

1. Right click mouse on the project in which you want add it. From drop-down menu click **Properties**

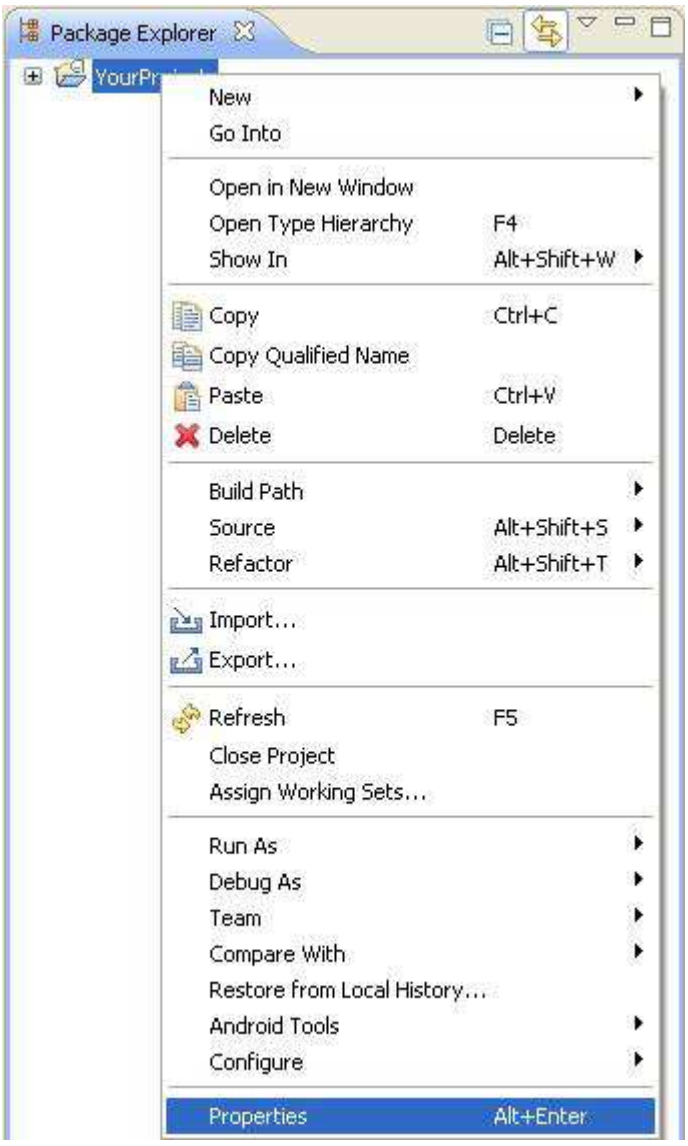


FIGURE 8 - PROJECT MENU ON ECLIPSE

2. Select **Java Build Path i**n the menu on the left

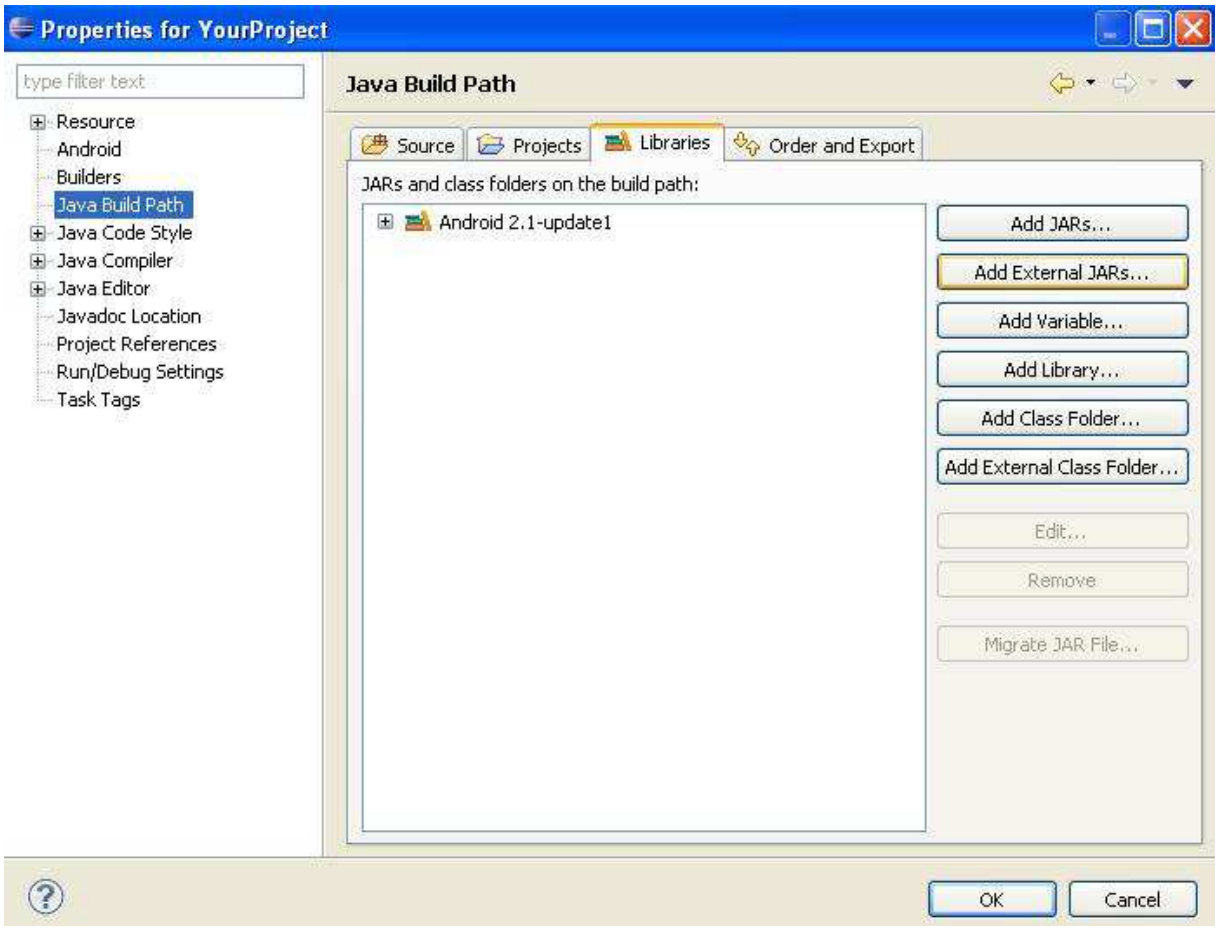


FIGURE 9 - PROJECT PROPERTIES WINDOW ON ECLIPSE

3. Click **Add External JARs…** and select the downloaded version of SCCT subscriber library indicating path.

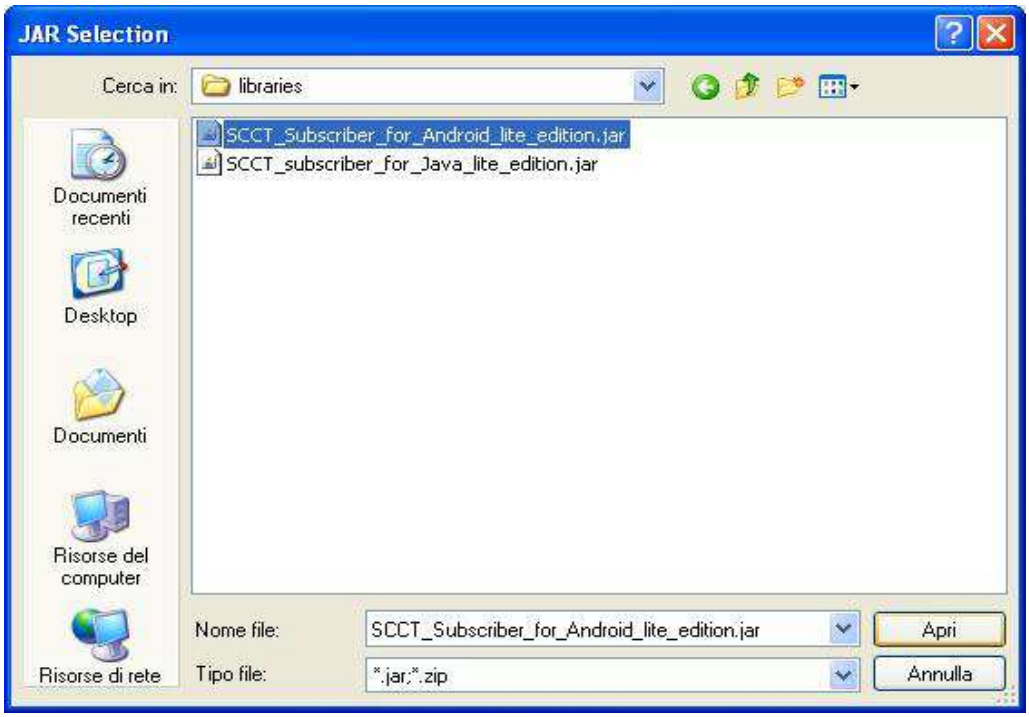
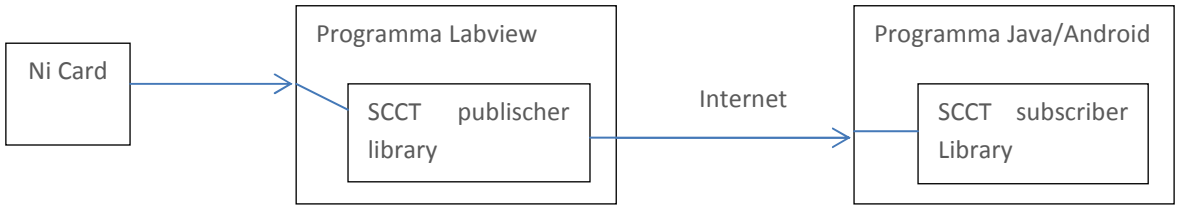


FIGURE 10 - JAR SELECTION WINDOW ON ECLIPSE

4. Click **OK** to close the window

GENERAL WORKING

SCCT subscriber library works together with SCCT publisher library that is developed in LabVIEW (to get details see www.toolsforsmartminds.com/products/SCCT.php). SCCT publisher library has the role of producer and give analog and digital data acquired from a remote system through the net to all clients who request it. SCCT subscriber library instead has the role of client, it receives data and provides it to the developer correctly and orderly, ready to be used yet. The developer only has to implement the interface provided by the library to receive data and use it.



SCCT Java Subscriber provides two fundamental classes for this work:

- Subject: has the task of handling direct the communication with the producer, provides some methods to manage the connection and provides received data to Observer objects.
- Observer : Provides an interface that has to be overridden by the developer to receive data.

Observer object must register itself to Subject if it want to receive data, and more Observer objects can be registered in the same Subject.

How SCCT Java subscriber works?

SCCT Java subscriber implements the Observer pattern. This is a pattern intuitively used as an architectural base of a lot of event management systems.

This pattern is substantially based on one or more objects, called observers or listeners, that are registered to manage an event that may be generated by the "observed" object.

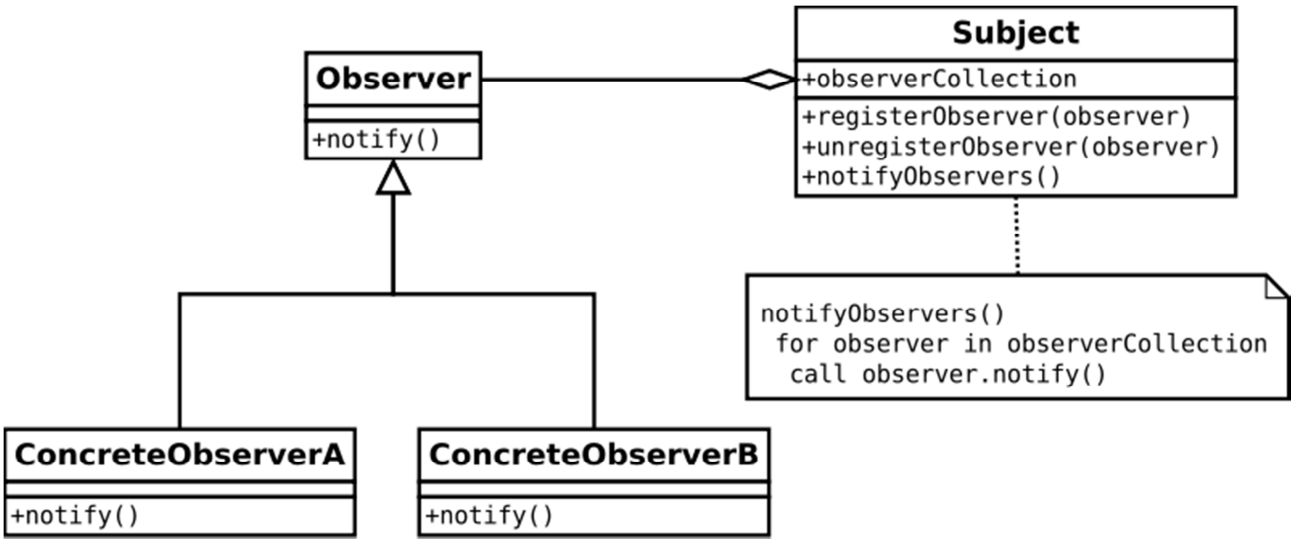


Figure 11 - Uml Scheme of Observer pattern

Uml Scheme of Observer pattern

The fundamental classes of this pattern are:

- Subject: Is a class that provides data (or packages) to observers and makes notifies calling the appropriate listeners.
- Observer: This class define an interface for all concrete observers, to receive data from Subject. It is an abstract class and must be implemented by ConcreteObservers.
- ConcreteObserver: it is a concrete implementation of Observer class. **This implementation must be done by developers.**

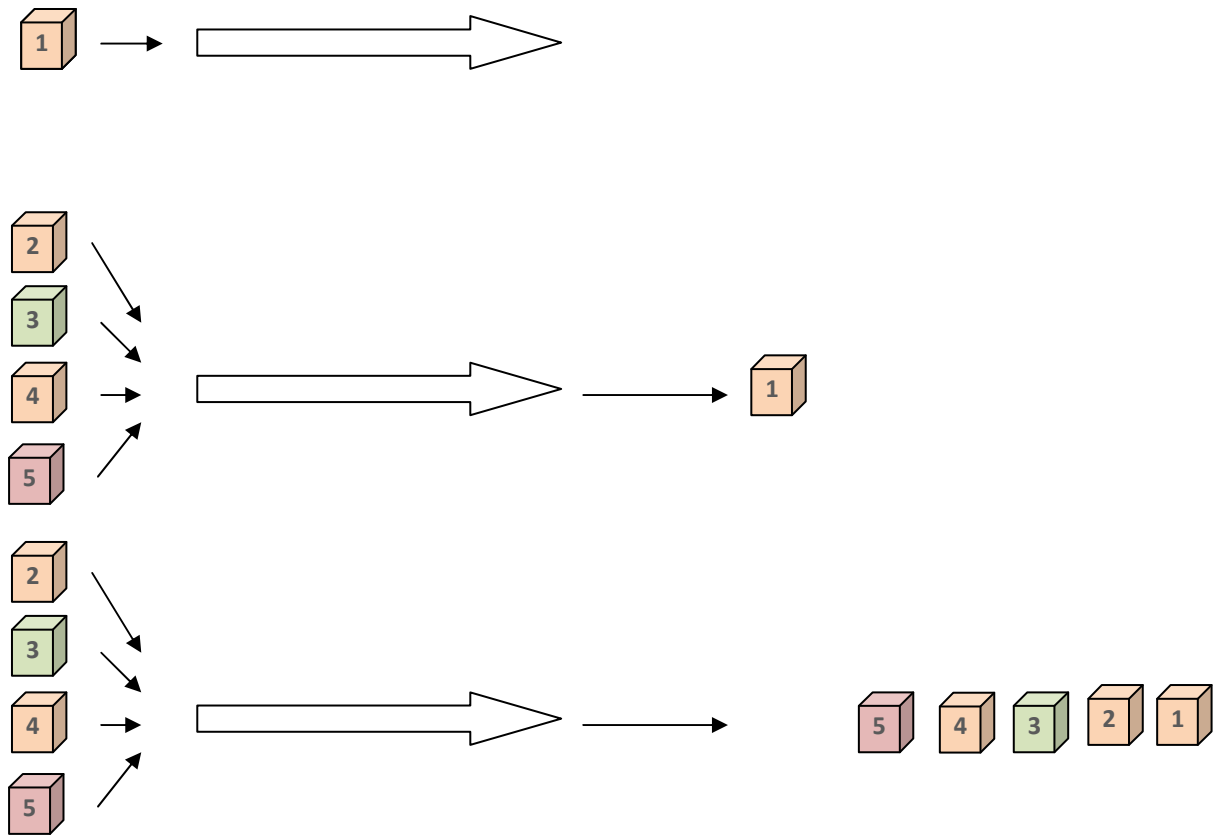
How the packages are managed

Each Observer puts the received packages in its own queue.

The packages are sorted by order of arrival as shown below.

Each Observer, later, extracts the first package from the queue and sends it in the appropriate listener.

If the listeners are too slow to process the packages, the queue size grows indefinitely.



PERMISSION

SCCT for Android needs internet connection to work. For using it you have to add in in the AndroidManifest the “android.permission.INTERNET” permission.

The Android Manifest should look like this:

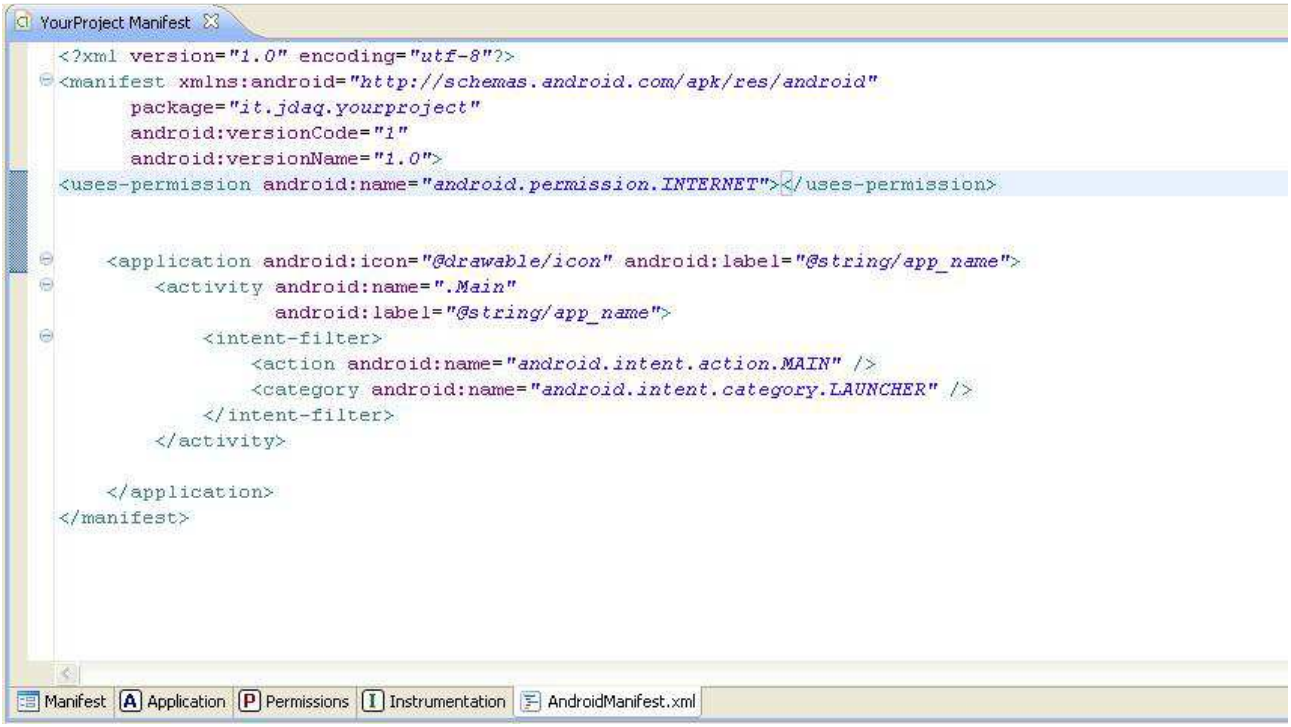


FIGURE 12 - ANDROIDMANIFEST

If you use saveFileOnExternalStorage() method of FilePackage class you have to add also “android.permission.WRITE_EXTERNAL_STORAGE” permission.

QUICK START

To start, you need to create the project and include the library as showed in the *Installation* chapter and you have to add permission how showed in *Permission* chapter .

Then you can start to create the ConcreteObserver class, as inner class or normal class. In this example, the normal class is used.

for android applications best design solution is creating ConcreteObserver as inner class because you can directly manage the elements of Activity.

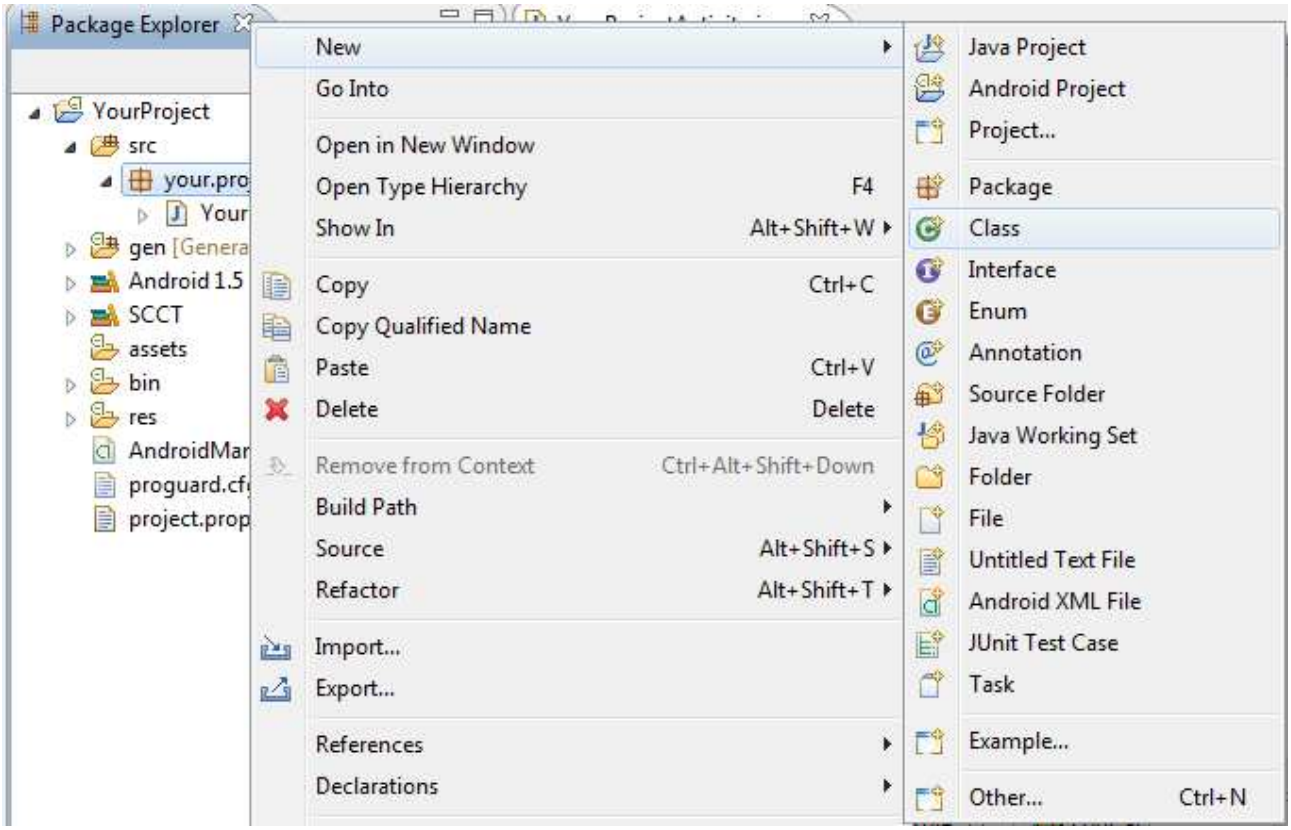


FIGURE 13 - NEW JAVA CLASS MENU ON ECLIPSE

Extend the ConcreteObserver with the SCCT_Observer class provided by the library.

```
public class ConcreteObserver extends SCCT_Observer {
```

FIGURE 14 - CONCRETEOBSERVER CLASS EXTENDS SCCT_OBSERVER CLASS

At the beginning the IDE returns an error. That's because SCCT.JavaSubscriber.SCCT_Observer is not included. The editor should provide a hint. If doesn't happen, you can add it manually:

```
package yourproject;

import SCCT.JavaSubscriber.SCCT_Observer;
```

FIGURE 15 - IMPORT SCCT.JAVASUBSCRIBER.SCCT_OBSERVER

Now the IDE will rises a new error: it means you haven't override the abstract methods of the SCCT_Observer class yet. Even in this case the IDE should provide a hint, creating itself the following code:

```
    @Override
    public void analogDataListener(AnalogDataPackage adp) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void digitalDataListener(DigitalDataPackage ddp) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void configurationListener(ConfigurationPackage cp) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void messageListener(MessagePackage ap) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void errorListener(ErrorPackage ep) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
```

FIGURE 16 - UNIMPLEMENTED METHODS OF SCCT_OBSERVER

Enter the following import statements below the package declaration:

```
import SCCT.JavaSubscriber.SCCT_Observer;
import SCCT.JavaSubscriber.AnalogDataPackage;
import SCCT.JavaSubscriber.ConfigurationPackage;
import SCCT.JavaSubscriber.DigitalDataPackage;
import SCCT.JavaSubscriber.ErrorPackage;
import SCCT.JavaSubscriber.MessagePackage;
```

FIGURE 17 - IMPORT SCCT CLASS

There, the Listeners are ready to be implemented. Inside every method we can write our code to manage events. Let's see an example:

```
@Override
public void analogDataListener(AnalogDataPackage adp) {
    Log.d("SCCT",adp.getCalendar().getTime().toString() + " - analog data received");
}

@Override
public void configurationListener(ConfigurationPackage cp) {
    Log.d("SCCT",cp.getCalendar().getTime().toString() + " - configuration received");
}

@Override
public void digitalDataListener(DigitalDataPackage ddp) {
    Log.d("SCCT",ddp.getCalendar().getTime().toString() + " - digital data received");
}

@Override
public void errorListener(ErrorPackage ep) {
    Log.d("SCCT",ep.getCalendar().getTime().toString() + " - error received");
}

@Override
public void messageListener(MessagePackage ap) {
    Log.d("SCCT",ap.getCalendar().getTime().toString() + " - message received");
}
```

FIGURE 18 - IMPLEMENTED METHODS OF SCCT_OBSERVER

The ConcreteObserver is ready. Let's see how implement the main activity class. Create an instance of SCCT_Subject class and another one of ConcreteObserver class.

```
public class YourProjectActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        SCCT_Subject subject = new SCCT_Subject();
        ConcreteObserver o = new ConcreteObserver();
    }
}
```

FIGURE 19 – MAIN ACTIVITY CLASS IMPLEMENTATION

You can import SCCT_Subject class.

```
package your.project;

import SCCT.JavaSubscriber.SCCT_Subject;
import android.app.Activity;
import android.os.Bundle;
```

FIGURE 20 - IMPORT SCCT_SUBJECT CLASS

Now you have to register the ConcreteObserver in the SCCT_ Subject object.

```
subject.registerObserver(o);
```

FIGURE 21 - REGISTER OBSERVER

Finally you have to open the communication and ask the data transmission's start.

```
try {
    subject.openCommunication("address", 8083, "ApiKey", "description");
    subject.startCommunication();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

FIGURE 22 - OPEN COMMUNICATION AND START DATA

That's done! If the parameters of the openCommunication method are correct, you should see something similar to this screen in your logCat:

L...	Time	PID	Application	Tag	Text
D	06-08 17:11:5...	655	your.project	SCCT	Fri Jun 08 19:11:52 GMT+00:00 2012 - configuration received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:51 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:51 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:52 GMT+00:00 2012 - analog data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:53 GMT+00:00 2012 - digital data received
D	06-08 17:11:5...	655	your.project	SCCT	Mon Jul 09 07:11:53 GMT+00:00 2012 - analog data received

FIGURE 23 - FINAL RESULT

ERRORS

The errors are generated when a problem occurs while connecting or if you set the wrong parameters in the openCommunication method.

Whenever an error occurs the connection is broken.

The library handles two type of errors.

The first type is generated before the communication's opening and is managed as an exception by the openCommunication, can be generated because one of these reasons:

Name	Description
Destination unreachable	This error is thrown if is impossible open the connection with the producer.
Empty address	This error is thrown if the address string is empty.
Timeout not correct	This error is thrown if the timeout is less than or equal to one.
Wrong port	This error is thrown if the port is less than or equal to zero.

The second type is managed as an ErrorPackage and it is sent to the errorListener. Each error is associated with a code (in the ErrorPackage class there are some constants to help the developer).

This error can be generated because of one of these reasons:

Code	Name	Description
1	Wrong API-Key	This error is thrown if API-Key is incorrect.
2	Wrong Timeout	This error is thrown if API-Key is incorrect.
3	Wrong timestamp	This error is thrown if the client's timestamp is too much different from the producer's one.
4	Expired timeout	This error is thrown if timeout is expired.
5	Transmission error	This error is thrown if it fails to send a message.

The last two errors are generated by the library and have a different timestamp from other packages (that are created by the producer).